#include <iostream>

#include <fstream>

#include <string>

#include <cassert>

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/csma-module.h"

#include "ns3/applications-module.h"

#include "ns3/internet-apps-module.h"

#include "ns3/internet-module.h"

#include "ns3/flow-monitor-module.h"

//#include "ns3/netanim-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("CsmaPingExample");

int

main (int argc, char \*argv[])

{

CommandLine cmd;

cmd.Parse (argc, argv);

Time interPacketInterval = Seconds (1.);

// Here, we will explicitly create four nodes.

NS\_LOG\_UNCOND ("Create nodes.");

NodeContainer c;

c.Create (6);

// connect all our nodes to a shared channel.

NS\_LOG\_UNCOND ("Build Topology.");

CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));

csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));

csma.SetDeviceAttribute ("EncapsulationMode", StringValue ("Llc"));

NetDeviceContainer devs = csma.Install (c);

// add an ip stack to all nodes.

NS\_LOG\_UNCOND ("Add ip stack.");

InternetStackHelper ipStack;

ipStack.Install (c);

// assign ip addresses

NS\_LOG\_UNCOND ("Assign ip addresses.");

Ipv4AddressHelper ip;

ip.SetBase ("192.168.1.0", "255.255.255.0");

Ipv4InterfaceContainer addresses = ip.Assign (devs);

NS\_LOG\_UNCOND ("Create Source");

InetSocketAddress dst = InetSocketAddress (addresses.GetAddress (3));

OnOffHelper onoff = OnOffHelper ("ns3::UdpSocketFactory", dst);

onoff.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

onoff.SetAttribute ("PacketSize", UintegerValue (1100));

onoff.SetAttribute ("DataRate", StringValue ("50Mbps"));

ApplicationContainer apps = onoff.Install (c.Get (0));

apps.Start (Seconds (1.0));

apps.Stop (Seconds (10.0));

NS\_LOG\_UNCOND ("Create Sink.");

PacketSinkHelper sink = PacketSinkHelper ("ns3::UdpSocketFactory", dst);

apps = sink.Install (c.Get (3));

apps.Start (Seconds (0.0));

apps.Stop (Seconds (11.0));

NS\_LOG\_UNCOND ("Create pinger");

V4PingHelper ping = V4PingHelper (addresses.GetAddress (0));

// ping.SetAttribute ("Interval", TimeValue (interPacketInterval));

ping.SetAttribute ("Interval", TimeValue (interPacketInterval));

NodeContainer pingers;

pingers.Add (c.Get (3));

pingers.Add (c.Get (1));

pingers.Add (c.Get (2));

pingers.Add (c.Get (4));

pingers.Add (c.Get (5));

apps = ping.Install (pingers);

apps.Start (Seconds (2.0));

apps.Stop (Seconds (10.0));

//Enable Tracing using flowmonitor

FlowMonitorHelper flowmon;

Ptr<FlowMonitor> monitor = flowmon.InstallAll();

Simulator::Stop (Seconds (10.0));

//Add visualization using Netanim

// AnimationInterface anim ("ex5.xml");

NS\_LOG\_UNCOND ("Run Simulation.");

Simulator::Run ();

// Print per flow statistics

monitor->CheckForLostPackets ();

Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());

std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

for (std::map<FlowId, FlowMonitor::FlowStats>::const\_iterator iter = stats.begin (); iter != stats.end

(); ++iter)

{

Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);

NS\_LOG\_UNCOND("Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst

Addr " << t.destinationAddress);

NS\_LOG\_UNCOND("Tx Packets = " << iter->second.txPackets);

NS\_LOG\_UNCOND("Rx Packets = " << iter->second.rxPackets);

std::cout << "Lost Packets = " << iter->second.lostPackets<< std::endl;

NS\_LOG\_UNCOND("Throughput: " << iter->second.rxBytes \* 8.0 / (iter-

>second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds()) / 1024 << "

Kbps");

}

Simulator::Destroy ();

NS\_LOG\_UNCOND ("Done.");

}